Shell Scripting

•••

Richard Huang

(content ~~stolen~~ based off the slides by Ian McDonald)

Course resources

- Your facilitators!
- All material (new & old) is online at decal.ocf.io
- Office hours + demos during the in-person lecture time - 8pm PST in ocf.io/decalzoom
- Email us or drop by in #decal-general via Slack, IRC, Discord, Matrix, etc.

Engaging with this lecture

- Follow slides online: ocf.io/decal/slides/b3
- Connect to login server:
  - `$OCF_USERNAME@ssh.ocf.berkeley.edu`
- Ask questions!
  - Yell at your screen
  - On #decal-general in Slack
  - Lectures are a lot more fun when you ask questions.

Topics

What's on the menu?

1. Bash
2. Variables
3. Conditionals
4. Loops
5. Functions
6. Streams

But why?

Good question

But why?

Good question

- You’re a sysadmin

But why?

Good question

- You’re a sysadmin
- You have to run some commands all the time

But why?

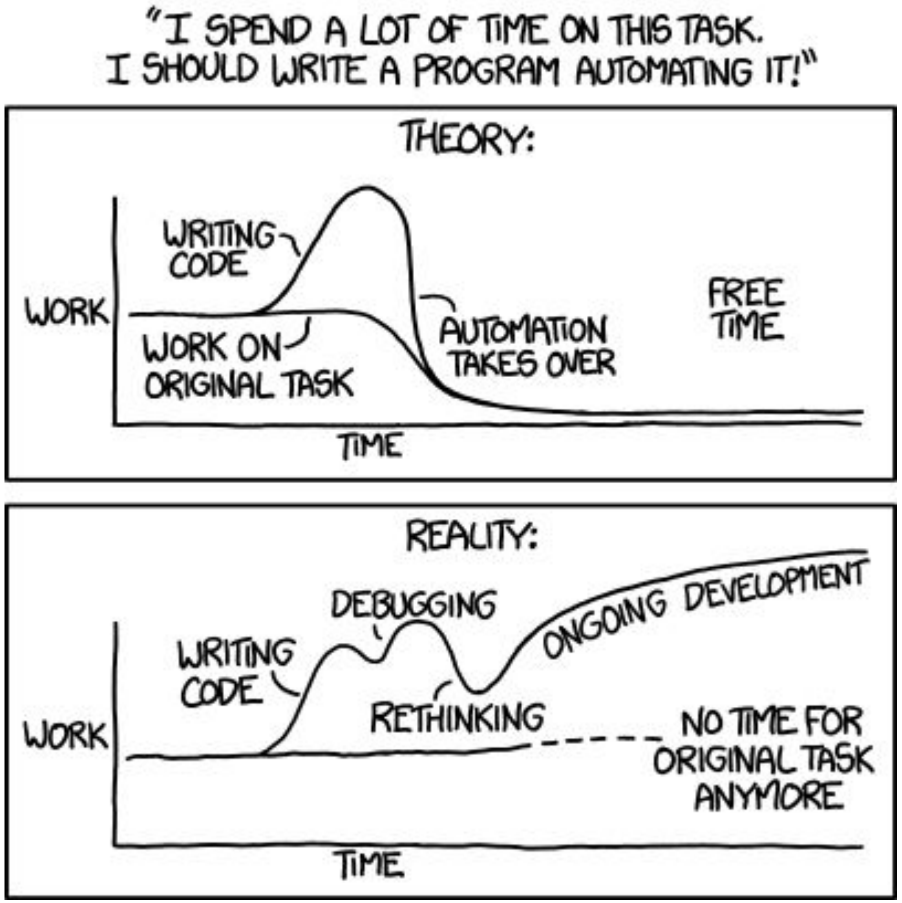
Good question

- You’re a sysadmin
- You have to run some commands all the time
- But you want to be ~~lazy~~ DRY

But why?

Good question

- You're a sysadmin
- You have to run some commands all the time
- But you want to be ~~lazy~~ DRY
- Describe your task as a step-by-step set of instructions so that a computer can do it for you!


"I SPEND A LOT OF TIME ON THIS TASK. I SHOULD WRITE A PROGRAM AUTOMATING IT!"
THEORY:
WRITING CODE
WORK
WORK ON ORIGINAL TASK
AUTOMATION TAKES OVER
FREE TIME
TIME
REALITY:
DEBUGGING
WRITING CODE
ONGOING DEVELOPMENT
RETHINKING
WORK
NO TIME FOR ORIGINAL TASK ANYMORE
TIME

Topics

What's on the menu?

1. **Bash**
2. Variables
3. Conditionals
4. Loops
5. Functions
6. Streams

Bash

A shell

Bash

A shell...

And also a programming language!

Comments

Use a pound/sharp/hashtag

```
# This is a comment
```

Shebang!

- Determines the program used to execute the lines below

```
#! /path/to/interpreter

#! /bin/bash

#! /bin/sh

#! /usr/bin/python
```

Running Your Script

- Remember to make your script executable

```
# Make executable
chmod +x your-script.sh

# Run script!
./your-script.sh
```

Topics

What's on the menu?

1. Bash
2. **Variables**
3. Conditionals
4. Loops
5. Functions
6. Streams

shell variables

- Whitespace matters!
- Variable interpolation with $
- Display text with `echo`

```
NAME="value"
echo "$NAME"
```

shell variables

- Types? What types?
- Bash variables are untyped

```
FOO=1
$FOO  +  1
```

shell variables

- Types? What types?
- Bash variables are untyped
- Operations are contextual

```
FOO=1
$FOO + 1
error!
```

shell variables

- Use the `expr` command to evaluate expressions

```
FOO=1
expr $FOO + 1
2
```

User input

- Use the `read` command get user input
- “-p” is for the optional prompt

```
read -p “send: ” FOO
# enter “hi”
echo “sent: $FOO”
sent: hi
```

subshell

- Command substitution allows you to use another command’s output to replace the text of the command

```
FOO=$(expr 1 + 1)
echo “$FOO”
2
```

Topics

What's on the menu?

1. Bash
2. Variables
3. **Conditionals**
4. Loops
5. Functions
6. Streams

test

test test mic check

- Evaluates an expression
- Also synonymous with [ ]
- Sets exit status to
    - 0 (true)
    - 1 (false)

test

test test mic check

- Evaluates an expression
- Also synonymous with [ ]
- Sets exit status to
  - 0 (true)
  - 1 (false)

(Yup you read that right)

test

Lots of comparators

`-eq`	`==`
`-ne`	`!=`
`-gt`	`>`
`-ge`	`>=`
`-lt`	`<`
`-le`	`<=`

test

Examples

```
test zero = zero; echo $?

test zero = one; echo $?
```

test

Examples

```
test zero = zero; echo $?
0 # 0 means true
test zero = one; echo $?
1 # 1 means false
```

test

Examples

```
[0 -eq 0]; echo $?
0 # 0 means true
[0 -eq 1]; echo $?
1 # 1 means false
```

if

What if...?

```
if [ “$1” -eq 69 ];
then
    echo “nice”
fi
```

if-else

…And what ifn’t

```
if [ “$1” -eq 69 ];
then
    echo “nice”
else
    echo “darn”
fi
```

elif

…And what ifn’t but if

```
if [ “$1” -eq 69 ];
then
   echo “nice”
elif [ “$1” -eq 42 ];
then
   echo “the answer!”
else
   echo “wat r numbers”
fi
```

case

No one likes long if statements

```
read -p "are you 21?" ANSWER
case "$ANSWER" in
  “yes”)
    echo "i give u cookie";;
  “no”)
    echo "thats illegal";;
  “are you?”)
    echo “lets not”;;
  *)
    echo "please answer"
esac
```

Topics

What's on the menu?

for loops

```
NAMES="a b c d"
for NAME in $NAMES
do
  echo "Hello $NAME"
done
```

while loops

```
while true
do
  echo "Hello $NAME"
done
```

Topics

What's on the menu?

1. Bash
2. Variables
3. Conditionals
4. Loops
5. **Functions**
6. Streams

functions

```
function greet() {
  echo “hey there $1”
}
greet “Richard”

hey there Richard
```

Topics

What's on the menu?

1. Bash
2. Variables
3. Conditionals
4. Loops
5. Functions
6. **Streams**

Redirection

Use **>** to output to a file

```
echo “hello” > file
```

Redirection

Use < to take input from a file

```
sort < file
```

Pipes

```
command1 | command2
```

Take output of first command and “pipe” it into the second one, connecting stdin and stdout

C’est n’est pas une pipe

Additional Notes

- Python
  - `argparse`: easy CLI
  - `fabric`: easy deployment
  - `salt`: generally useful for infrastructure-related tasks
  - `psutil`: monitor system info
- Use **bash** when the functionality you want is easily expressed as a composition of command line tools
  - Common file manipulation operations
- Use **Python** when you need "heavy lifting" with complex control structures, messy state, recursion, OOP, etc.

Other Resources

- AT&T Archives: The UNIX Operating System
- Knuth and McIlroy Word Count
- Linux Documentation Project: Bash Guide for Beginners
- Honestly, Google is your best friend